



# CHAPTER 14

## TUPLES, SETS, AND DICTIONARIES

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING USING PYTHON, LIANG (PEARSON 2013)

# MOTIVATIONS

- How would we define a movie with its title and year?
  - Normally, we make an extensive class, but this might be overkill.
  - Here, we can use tuples
- What about a No-Fly-List to screen individuals who are banned from travel?
  - We could maintain the list, but it will be inefficient to work with and operate on.
  - Here, we can use sets
- What if we wanted to store student records and access them by student ID?
  - Again we could maintain a list, but this will be inefficient
  - Here, we can use dictionaries



# TUPLES

- **Tuples** are like lists except they are immutable. Once they are created, their contents cannot be changed.
  - Almost every operation that can be performed on a list can be performed on a tuple
- If the contents of a list in your application do not change, you should use a tuple to prevent data from being modified accidentally.
- Tuples are the magic behind returning more than one thing from a function
- Furthermore, tuples are more efficient than lists.

# CREATING TUPLES

- There are various ways you can create a tuple, including:
  - Creation of an empty tuple, or a tuple from a series of elements using `()` (not `[]`)
    - `t1 = ()` # Create an empty tuple
    - `t2 = (1, 3, 5)` # Create a tuple with three elements
  - Creating a tuple from other types, e.g., lists or strings
    - # Create a tuple from a list  
`t3 = tuple([2 * x for x in range(1, 5)])`
    - # Create a tuple from a string  
`t4 = tuple("abac")` # t4 is ['a', 'b', 'a', 'c']

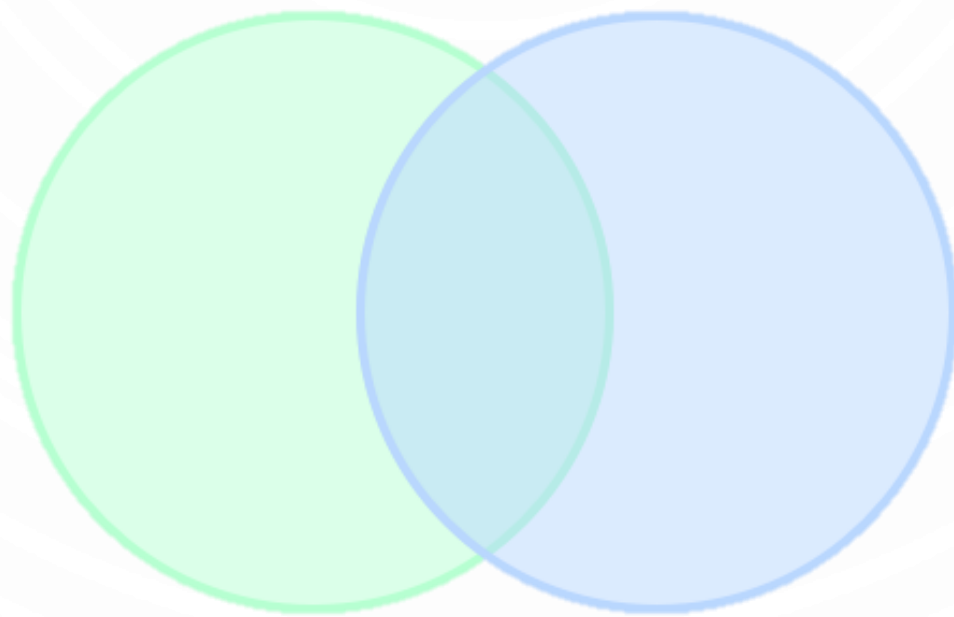
# SETS

- **Sets** are like lists and store a collection of items.
  - Most operations that can be performed on a list can be performed on a set, but with some slight semantical differences
- Unlike lists, the elements in a set are *unique* and are *not placed in any particular order*.
  - If your application does not care about the order of the elements, using a set to store elements is more efficient than using lists.
- The syntax for sets is braces { }.

# CREATING SETS

- There are various ways you can create a set, including:
  - Creation of an empty set, or a set from a series of elements using `{}` (not `[]`)
    - `s1 = set()` # Create an empty set
    - `s2 = {1, 3, 5}` # Create a set with three elements
  - Creating a set from other types, e.g., lists or strings
    - # Create a set from a list  
`s3 = set([2 * x for x in range(1, 10)])`
    - # Create a set from a string  
`s4 = set("abac")` # s4 is {'a', 'b', 'c'}

# MATHEMATICAL SETS



# OPERATIONS WITH SETS

- The method `s1.issubset(s2)` will determine if `s1` is a subset of `s2`, similarly there is a method `issuperset`.

- `s1 = {1, 2, 4}`

- `s2 = {1, 4, 5, 2, 6}`

- `s1.issubset(s2) # True, as s1 is a subset of s2`

- Equality test between two sets returns true if all of the same contents exist between them

- `s1 = {1, 2, 4}`

- `s2 = {1, 4, 2}`

- `s1 == s2 # True`



# SET COMPARISON OPERATORS

- It makes no sense to compare sets using the conventional comparison operators ( $>$ ,  $>=$ ,  $<=$ ,  $<$ ), because the elements in a set are not ordered. However, these operators have special meaning when used for sets.
  - $s1 > s2$  - returns true means  $s1$  is a proper superset of  $s2$ .
  - $s1 >= s2$  - returns true means  $s1$  is a superset of  $s2$ .
  - $s1 < s2$  - returns true means  $s1$  is a proper subset of  $s2$ .
  - $s1 <= s2$  - returns true means  $s1$  is a subset of  $s2$ .

# SET UNION

- Consider:

$s1 = \{1, 2, 4\}$

$s2 = \{1, 3, 5\}$

- **Union** (or `|` operator) between two sets retains all elements between them

`s1.union(s2)` #  $\{1, 2, 3, 4, 5\}$

`s1 | s2` #  $\{1, 2, 3, 4, 5\}$

# SET INTERSECTION

- Consider:

$s1 = \{1, 2, 4\}$

$s2 = \{1, 3, 5\}$

- **Intersection** (or & operator) between two sets retains only elements in common between the two sets

```
s1.intersection(s2) # {1}
```

```
s1 & s2             # {1}
```

# SET DIFFERENCE

- Consider:

`s1 = {1, 2, 4}`

`s2 = {1, 3, 5}`

- **Difference** (or `-` operator) between two sets retains elements in the first but not in the second

`s1.difference(s2) # {2, 4}`

`s1 - s2 # {2, 4}`

# SET SYMMETRIC DIFFERENCE

- Consider:

$s1 = \{1, 2, 4\}$

$s2 = \{1, 3, 5\}$

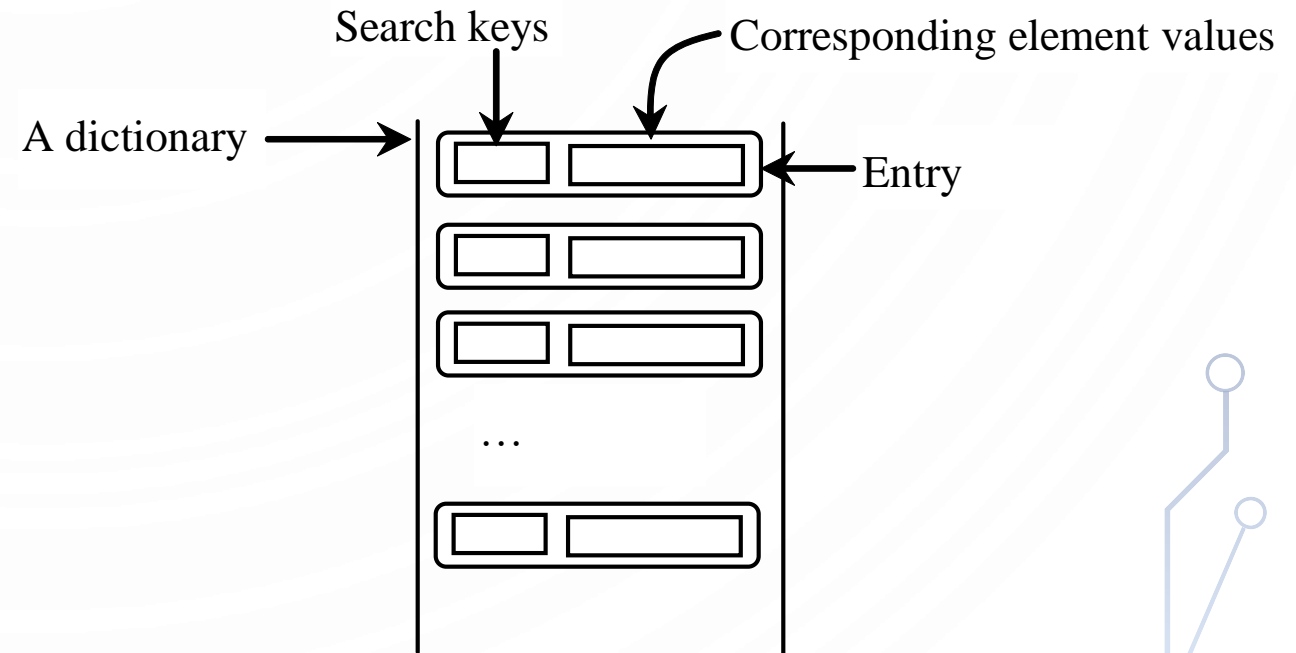
- **Symmetric Difference** (or  $\wedge$  operator) between two sets retains only elements which exist either in one or the other, but not both

`s1.symmetric_difference(s2)` # {2, 3, 4, 5}

`s1 ^ s2` # {2, 3, 4, 5}

# DICTIONARY

- A dictionary is a collection of key, value pairs. The key is like a name of the element that allows quick access to it.
  - From our motivating example of a student record – the key is a student ID and the entire student data is the value



# CREATING A DICTIONARY

- Again there are various ways to make a dictionary:
  - `d1 = {}` # Create an empty dictionary
  - `d2 = {"john":40, "peter":45}` # Create a dictionary
- When listing the elements, the first literal is a key and the second literal is the value (separated by a :)

# ADDING/MODIFYING ENTRIES

- To add or modify an entry to a dictionary:

- `dictionary[key] = value`

- For example:

- `d2["susan"] = 50`



# DELETING ENTRIES

- To delete an entry from a dictionary:

- `del dictionary[key]`

- For example:

- `del d2["susan"]`

# LOOPING OVER ENTRIES

- A for loop over a dictionary will loop over its keys. As an example:

```
for key in dictionary:  
    print(key + ":" + str(dictionary[key]))
```

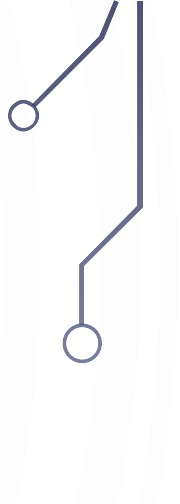

# OPERATIONS WITH DICTIONARIES

- Similar operations exist for dictionaries as did other data structures
  - `len(dict)` counts the number of entries into the dictionary
  - `in/not in` tests existence of keys
- Other methods:

dict	
<code>keys(): tuple</code>	Returns a sequence of keys.
<code>values(): tuple</code>	Returns a sequence of values.
<code>items(): tuple</code>	Returns a sequence of tuples (key, value).
<code>clear(): void</code>	Deletes all entries.
<code>get(key): value</code>	Returns the value for the key.
<code>pop(key): value</code>	Removes the entry for the key and returns its value.
<code>popitem(): tuple</code>	Returns a randomly-selected key/value pair as a tuple and removes the selected entry.



# SUMMARY

- **Tuples** – immutable lists
  - **Sets** – collection of unique, unordered elements
  - **Dictionaries** – collection of key-value entries
- 
- 
- 